Consider the following database schema :

- Instructor(*InstNo*, Name, Rank)
- Student(*StudentNo*, Name, Address, EnrollD, GraduatD)
- Course(*CourseNo*, Title, nbHours, InstNo)
  InstNo references Instructor(InstNo)
- Take(*StudentNo*, *CourseNo*, Grade)
  StudentNo references Student(StudentNo)
  CourseNo references Course(CourseNo)

Exercise 1
Write the following transactions. What is the output ? Comment !

(a)

```
begin transaction
insert into Instructor values ('1111','Lionel', 'Full', 30)
select * from Instructor
rollback
select * from Instructor
```

(b)

```
begin transaction
insert into Instructor values ('1111','Lionel', 'Full', 30)
select * from Instructor
commit
select * from Instructor
```

(c)

```
begin transaction
insert into Instructor values (null,'Lionel', 'Full', 30)
select * from Instructor
commit
select * from Instructor
```

Exercise 2
Write a transaction that does the following actions :

(a) Insert a new instructor

(b) Associate all courses taught by instructor number 12345 to the inserted instructor

(c) Delete the instructor number 12345.

Solution

```
begin tran
insert into Instructor values ('54321', 'Lionel','Full',25)
update Course set InstNo='54321' where InstNo='12345'
delete from Instructor where InstNo='12345'
commit
```

Exercise 3

(a) Make a copy named "Temp" (structure + data) of the Student table

(b) Display the content of the Temp table

(c) Write a transaction that deletes all rows from the table "Temp"

(d) Check if the table "Temp" is empty

(e) Cancel the last operation DELETE

(f) Check that all rows before the transaction exist

Solution

```
select *
into Temp
from Student

SELECT * FROM Temp;

BEGIN TRANSACTION;
DELETE FROM Temp;
insert into Temp values ('02', 'sara','erzey',GETDATE(),GETDATE(), 20)
SELECT * FROM Temp;
ROLLBACK;

SELECT * FROM Temp;
```

Pour copier la structure et les données d'une table table1 vers une nouvelle table table2 : SELECT * INTO table2 FROM table1 ; GO
Pour copier uniquement la structure d'une table1 vers une nouvelle table table2 : SELECT * INTO table2 FROM table1 WHERE 1=0 ; GO

Exercise 4
Write the following transaction :

(a) Add a new instructor

(b) Add a first course for the inserted instructor

(c) Add a second course for the inserted instructor

Test the case where this transaction is committed and the case where this transaction is rolled back.
Solution

```
begin tran
insert into Instructor values ('bilal', 'bilal','Full',25)
if @@ERROR=0
    begin
    insert into Course values ('INFOBi','Bilal', 50,'bilal')
    if @@ERROR=0
    begin
        insert into Course values ('INFOBi1','Bilal', 50,'bilal')
        if @@ERROR =0
            commit tran
        else
            rollback tran
    end
    else
        rollback tran
    end
else
    rollback tran
```

Exercise 5
Write the following constraints using triggers.

(a) The age of students must be greater than 18
Solution

```
Using a CHECK constraint
-------------------------------
alter table Student
add constraint student_Age18
check (dateadd(year,18,BDate) <= getdate())

Using a trigger for insert, same for update
-------------------------------
create trigger age18
on Student
after insert
as
if exists(
select *
from Inserted
where dateadd(year,18,BDate) > getdate() )
begin
delete from Instructor where InstNo = (Select InstNo from inserted)
raiserror ('Violation, age must be greater than 18', 15 , 120)
end
```

(b) The instructor must be older than all his students
Solution

```
create trigger instructorAge
on Instructor
after insert
as
if exists(
select *
from Inserted I, Student S, Course C, Take T
where(I.InstNo = C.InstNo and C.CourseNo=T.CourseNo and T.StudentNo=S.StudentNo and I.
    age > S.age))
begin
raiserror('Constraint Violation: The age of an instructor must be less than the age of
    his students', 15, 120)
delete from Instructor where InstNo=(select InstNo from inserted)
end
```

(c) An instructor teaches at most 4 courses
Solution

```
alter rigger insNbrCourses
on Course
after insert, update
as
if exists(
select *
from Course C
group by C.InstNo
having count(*) > 3)
begin
raiserror('Constraint Violation: Instructor must have at most 4 courses', 15, 120)
delete from Course where CourseNo=(select CourseNo from inserted)
end
```

Exercise 6
Write the following triggers :

(a) A trigger that prohibits the modification of grades
Solution

```
-- using after trigger ----------------
alter TRIGGER modifyGradeNotAllowed
   ON Take
   after update
AS
BEGIN
    SET NOCOUNT off;

    if(UPDATE(Grade))
     begin
     raiserror('Grades could not be modified', 15, 120)
     update Take
     set Grade = (select Grade from deleted)
     where StudentNo= (select studentNo from inserted) and CourseNo=(select CourseNo
         from inserted)
     end
END
GO


-- using instead of trigger ----------------
disable trigger modifyGradeNotAllowed on Take
go

alter TRIGGER modifyGradeNotAllowedInstead
   ON Take
   instead of update
AS
BEGIN
    if(UPDATE(Grade))
            raiserror('Grades could not be modified', 15, 120)
        else
        begin
            update Take
            set Grade = (select Grade from inserted)
            where StudentNo= (select studentNo from inserted) and CourseNo=(select
                CourseNo from inserted)
      end
END
GO
```

(b) A trigger that prevents any operation on the table instructor

(c) A trigger that after removing an instructor, remove all its related information

(d) Write a trigger that rolls back when trying to add an instructor that is already inserted.

(e) A trigger that prohibits to decrease the number of course hours.

Exercise 7
Create a table that has two columns : id and name. Make the id auto-increment using triggers. The initial seed should be 10.
Solution

```
create table trig(
id int,
name varchar(15)
)

ALTER TRIGGER autoInc
   ON  trig
   after insert
AS
declare @id int
```

```
declare @newId int
declare @t nvarchar(50)
declare @c int

select @t = name from inserted
select @id = id from inserted
delete from trig where id=@id

select @c = count(*) from trig


if @c=0 -- is Empty
    insert into trig values(10,@t)
else -- not empty
   begin
   select @newID = max(id) from trig
   insert into trig values(@newID+1, @t)
   end
go
```

Exercise 8
One wants to very quickly retrieve some statistics for each student and for this one needs a 'master table' : StatStudent (id, nbS, nbF, av). This table maintains the number of successful courses (nbS), the number of failed courses nbF and the grade average (av) of each student. Write a trigger that maintains the table StatStudent updated correctly whenever the table "Takes" is updated. Make sure you can handle every possible update of "Takes" that influences StatStudent.

Exercise 9
Create an audit trail that stores changes in the table "Take".